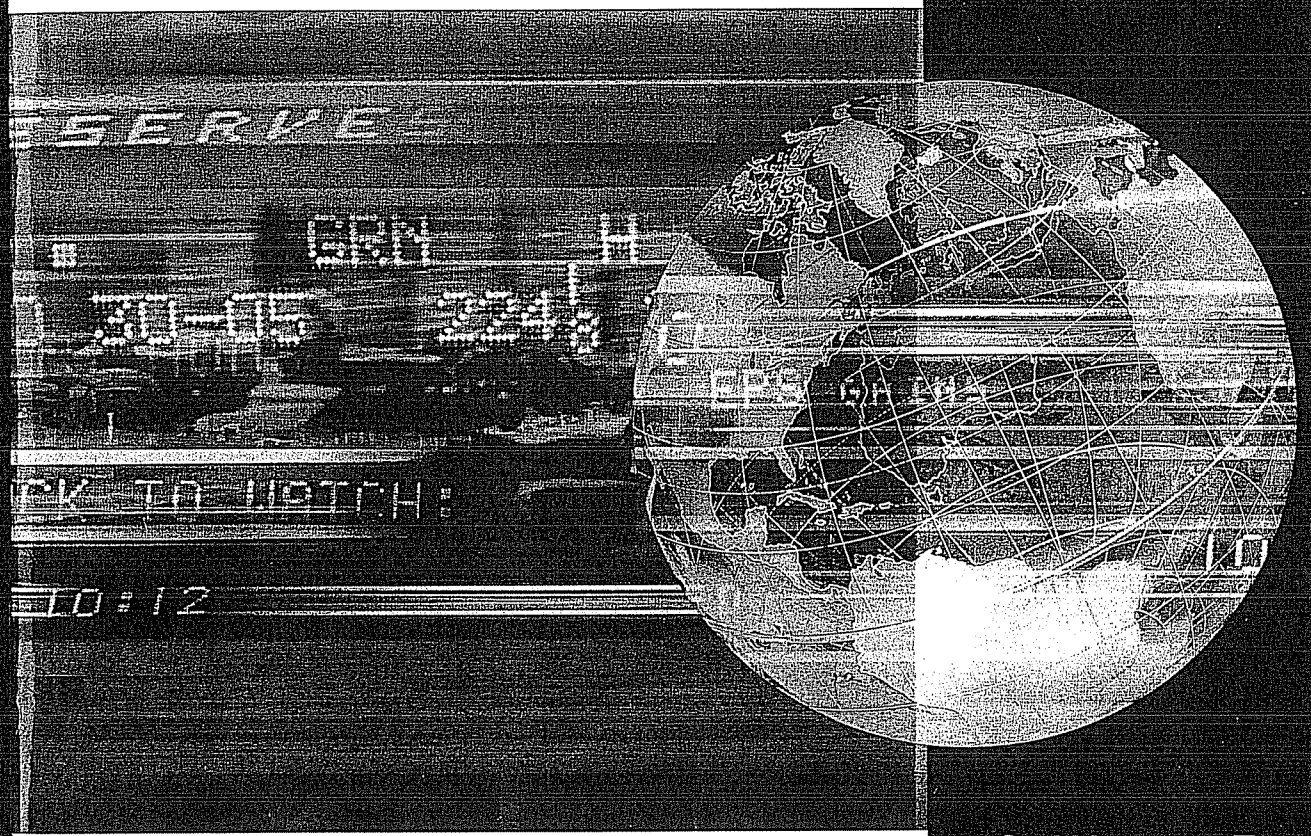
EXHIBIT E

Designing Systems for Internet Commerce

Second Edition



G. Winfield Treese
Lawrence C. Stewart

Designing Systems for Internet Commerce

Second Edition

G. Winfield Treese
Lawrence C. Stewart

Addison-Wesley

Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

The publisher offers discounts on this book when ordered in quantity for bulk purchases and special sales. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact:

International Sales
(317) 581-3793
international@pearsontechgroup.com

Visit Addison-Wesley on the Web: www.awprofessional.com

Library of Congress Cataloging-in-Publication Data

Treese, G. Winfield.
Designing systems for Internet commerce / G. Winfield Treese, Lawrence C. Stewart.—2nd ed.
p. cm.
Includes bibliographical references and index.
ISBN 0-201-76035-5 (pbk. : alk. paper)
1. System design. 2. Electronic commerce. I. Stewart, Lawrence C. II. Title.

QA76.9.S88 T74 2003
658.8'00285'4678—dc21

2002074788

ISBN: 0-201-76035-5
Text printed on recycled paper
1 2 3 4 5 6 7 8 9 10—CRS—0605040302
First printing, September 2002

signed by its authoring organization (digital signatures are described in Chapter 13). Before running the control, the user decides whether or not to trust the organization that created it.

ActiveX controls are very powerful, but they contain binary computer code, making them dependent on particular platforms. At this time, ActiveX is restricted to various versions of Microsoft Windows. In addition, the flow of a user's experience at a Web site may be disrupted when an ActiveX control is installed if the user must interrupt the process to answer a question about trusting the creator of the ActiveX control. ActiveX currently has no provision for removing controls once they are installed, so frivolous or seldom-used controls will accumulate and consume the user's system resources.

Sessions and Cookies

The HTTP protocol is designed to be stateless. Each request is intended to be independent of every other request. As the Web has come to be used as the foundation for complex applications (for commerce and other areas), cookies and other technologies have been developed to maintain persistent application state on top of the stateless protocol.

Why Sessions Are Important

As originally conceived, the Web was a very large collection of documents. Browsers would request a document, the user would work with it for a while, and then the user would request another document. In this environment, a stateless protocol makes sense. With many browsers and few servers, it is appropriate to make the server stateless so that it uses few resources per request. Today, however, almost any interesting Web-based application, particularly an Internet commerce application, requires a whole series of actions by the user and the server, working through a number of different Web pages. Therefore, it is important to treat such a series of actions as a single session of work.

State and Sessions

What is really going on in an application that has a series of interactions with the user? At the most basic level, the application has to remember state, and to make changes to the stored state as a result of interactions with the user. The situation is also complicated by the fact that there may be many users whose interactions with the Web application are interleaved with one another. Sessions are a way to remember which state information is associated with which user.

The way to approach this problem is to consider the possible places to keep application state. The general context is shown in Figure 9-1. A Web browser communicates

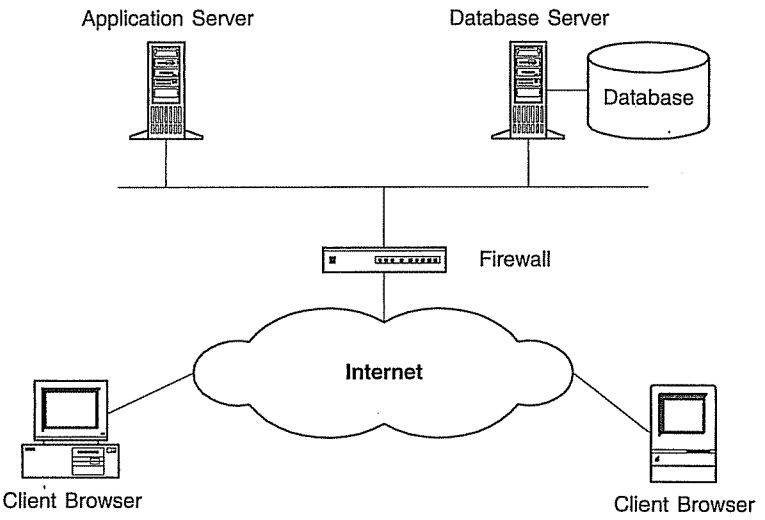


FIGURE 9-1. A Generic Web Application

with an application running on a Web server, which may also use a database. The state of the application can be kept in the database, in the application, or in the client.

State Kept in Database

In this model, the application itself is stateless, perhaps implemented as a CGI program that is executed anew on every client request. If the application requires a series of interactions with the user, intermediate application state is stored in the database. On each client request, it is necessary to locate and read the state from the database. This can be done by using Web basic authentication, which supplies a username and password on each request, or it can be done with a cookie or other session mechanism (discussed later).

State Kept in Application

If the application intermediate state is substantial, and the intermediate state does not require transaction safety, it may be kept in allocated storage inside the application. In this case, the application must be persistent (not restarted between client requests), as is possible with FastCGI, ISAPI, and NSAPI. Again, a session mechanism must be used to associate the particular user with a particular block of stored state information.

State Kept in Client

For simple applications, it may be appropriate to keep application state in the client between requests. In this model, when the application server responds to a client request with some intermediate results, it also sends to the client any intermediate application state that will be necessary to resume the application on the user's next request. This requires a mechanism for handing state information back and forth between client and server.

Session and Client State Mechanisms

There are several techniques for identifying a set of user requests as belonging to the same session and for passing state information back and forth between client and server. In addition, it is important to note that the difference between a session mechanism and a state mechanism is subtle. State is the application information itself, whereas a session identifier is a reference to state stored somewhere else.

Authentication Mechanisms

Any authentication mechanism, such as basic authentication or client certificates, can serve as a session mechanism. Because these authentication systems identify the user to the server on each request, state information can be stored at the application or application database and looked up on each request by using the user identification.

Dynamic URLs

If a user ID or other information is embedded in every URL used for a sequence of pages, the URLs in use by a particular user will be different from the URLs used by any other user. This creates a session mechanism. The difficulty is that these dynamic URLs can be easily lost if the user clicks out to some other Web site for a while and then reenters the application later. In addition, dynamic URLs can present a security challenge unless they are cryptographically protected from forgery or tampering.

Cookies

Most of the current generation of browsers support cookies, an idea originally introduced by Netscape. A cookie is a block of information transmitted from server to browser and stored there. On subsequent requests to the server, the browser sends back the cookie along with the Web request. Cookies can be used to store session identifiers or to store application state. In addition, cookies can be set to be persistent across browser sessions. The privacy implications of cookies are discussed further in Privacy Versus Merchandising, which begins on page 77. Because cookies are accessible to the savvy browser user, they must be protected from tampering.

Forms

If the Web application consists of a series of HTML forms, application state or session identification can travel along with the form as a hidden field. A hidden field is a special type of HTML form field that is not displayed to the user but merely returned as is to the server when the form is submitted. Like dynamic URLs and cookies, hidden fields are subject to tampering if not protected.

Applets

As Web applications evolve from simple HTML with forms and become more like client-server systems, it will become more common to use applets, controls, and client scripting to store session identifiers or application state.

Object Technology

Object technology has become the subject of intense marketing, as competing companies apply the label to whatever they were doing anyway. Consequently, it is necessary to dig fairly deep to understand exactly what is going on. The essential idea of a software object is a package containing both some *data* and the *methods* that operate on that data. This packaging has several good results.

- Encapsulation—objects hide the details of their implementation.

  Objects are a step beyond subroutine libraries, or Application Programming Interfaces (APIs). The programmer who develops software using APIs must understand how data is represented and stored. With object technology, the object is responsible for the data, not the programmer making use of the object to accomplish some higher goal. Encapsulation not only makes it unnecessary for the programmer to learn about the internal details of the object, it makes it impossible. This sounds heavy-handed, but it greatly reduces both immediate problems and latent bugs.

- Polymorphism—there can be multiple implementations of an object.

  Once a particular concept, such as a bank account, has been represented as an object, variations of the concept, such as a checking account or a savings account, can be represented as different implementations of an object, each with the same interface. When this is done, applications that operate on the account, such as the application that computes interest due each month, do not even need to know that there are different kinds of accounts.

- Language binding—objects can be implemented in different languages.

  The internal communications between one part of a program and another part are usually orchestrated by the language and its compiler. However, because the object technology carefully defines the ways in which one object can call another, there is

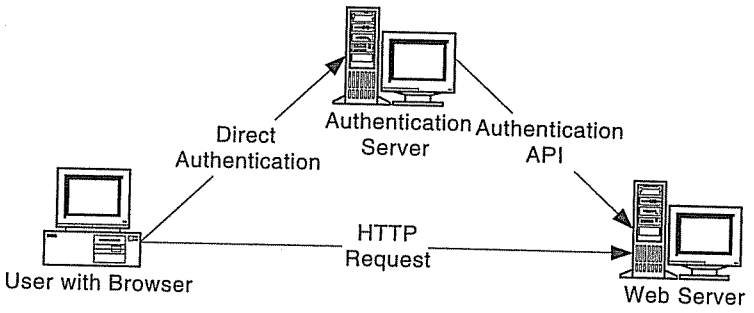


FIGURE 14-5. **Indirect Authentication**

and the lack of deployed smart card readers on PCs. One area in which hardware solutions may arise first is in wireless, where many new wireless devices include readers for smart cards. Figure 14-4 shows some alternatives for the storage of a user's authentication credentials.

Indirect Authentication

Not all Web authentication occurs directly between the user and the Web server that requires authentication. For indirect authentication, as illustrated in Figure 14-5, the Web server hands off the problem to a remote authentication server. There are several software products that implement third-party authentication, such as those from Netegrity and RSA Security, and there are network service versions, such as Microsoft Passport and the work underway at the Liberty Alliance. The authentication server of a third-party system in turn typically uses one of the direct authentication techniques described above.

Server Authentication

The weakest method of server authentication is to depend on the integrity of the Domain Name System. When the user types *www.xyzcorp.com* into the browser, there is a good chance that the connection will be made to the right server. However, this depends on correct management of many computers, together with constant vigilance by their operators, so stronger methods are desirable.

The most effective means of server authentication available today is the public-key certificate. Secure Web protocols such as SSL permit the server to deliver to the client a public-key certificate signed by a third party—the certification authority (CA). This method requires that the user's browser has the root key for the CA and that SSL/TLS be used for the communications. If the connection succeeds, the user knows that the CA has verified the identity of the server.

In a commercial context this verification may not be adequate, however. Suppose that the user wants to do business with XYZ Corporation, which has an online store. The Web pages for this store may be hosted on a shared computer together with pages from other companies. Consequently, the server certificate may read "Joe's Hosting Service" rather than "XYZ Corporation," which has a good chance of confusing the user. Unfortunately, there is little to be done about this problem.

This problem was handled in the little-used S-HTTP protocol, because every hyperlink in the Web could contain information about the security properties expected of the destination page. Once the user was satisfied with the initial page, the security properties of subsequent pages would be handled automatically.

Web Sessions

As mentioned earlier, the basic Web protocols are stateless, and each request stands on its own. Consequently, each request must be independently authenticated. This is exactly what happens in basic authentication, in which the user's name and password are passed to the server on each request. The user experience is not bad, because after the first request for name and password from a site, the browser remembers to supply the credentials on subsequent requests. However, the server must independently validate the name and password on every hit. In addition, although the use of a secure protocol such as SSL can overcome the problem of transmitting passwords in the clear, these protocols may not be warranted for all communications, because the content being viewed may not be intrinsically valuable. Digest authentication can solve these problems by making the authentication both secure and lightweight, but it is not widely deployed.

The technique most often used to enable authentication without requiring that all content be encrypted is to create a *session* on top of the basic stateless Web protocols. When the session is entered, the server first authenticates the user. Thereafter, the authentication information is not required on every request, but every request does include information that identifies it as belonging to a particular session. There are two ways to create a Web session today: custom URLs and cookies, as shown in Figure 14-6.

In the custom URL method, the identification of the session is carried in the URL, either as part of the URL query string

http://www.xyz.com/url/path/name/script.cgi?query&string&with&session_ID

or buried in the URL

http://www.xyz.com/<sessionId>/url/path/name

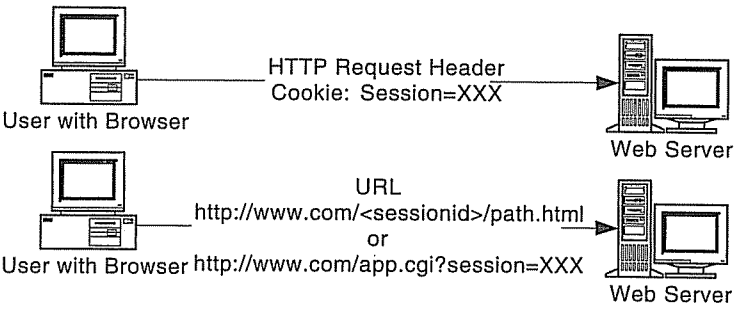


FIGURE 14-6. Session Management

The former approach requires that all user interaction be handled by a particular server application, such as a CGI program, that manages the session, and the latter approach requires that the session validation be built into the Web server itself.

In the cookie approach, the session identification information is stored in the browser cookie, so that the browser automatically presents it to the server on every request. This works very well, but there are several problems: not every browser supports cookies, Web proxy servers do not always handle them correctly, and some users may configure their browsers to reject cookies. Even when the mechanism works, if the client communications are not encrypted, the cookie may be stolen by an eavesdropper, who may then use it to gain illegitimate access to the Web site.

For encrypted communications, SSL version 3 and TLS offer an interesting alternative: the client certificate may be requested on the first request to a site. Thereafter, SSL session key caching can be used to reduce the average cost of the authentication by applying it across multiple requests. Both SSL version 3 and TLS offer the added capability of authentication-only connections, which do not encrypt the content or incur the associated performance penalty.

Summary

We have argued that security needs to be a property of the entire system. The essential problem of the security officer is that security has to be strong everywhere, because the attacker needs to find only one weak spot or lapse of operational attention. Security cannot be obtained by use of cryptographic pixie dust alone, although the lack of such technology can leave a system insecure. We advocate a five-step cycle of effort in security: creation of a security policy, addition of security mechanisms and technol-

ogies, careful design and operation of the computer and network environment, monitoring and auditing of the operational system, and, finally, evaluation of operations in order to refine the design, implementation, and operation of the system. The security design of the system must also incorporate principles of containment: security problems in one area of the system should be prevented from spreading to other areas.